

# node.js

A quick tour

by Felix Geisendörfer

# Introduction

# Why?

Node's goal is to provide an easy way to build scalable network programs.

-- [nodejs.org](http://nodejs.org)

# How?

Keep slow operations from blocking other operations.

# Traditional I/O

```
var data = file.read('file.txt');  
doSomethingWith(data);
```

Something is not right here

# Traditional I/O

```
var data = file.read('file.txt');  
// zzzZZzzz FAIL!  
doSomethingWith(data);
```

Don't waste those cycles!

# Async I/O

```
file.read('file.txt', function(data) {  
  doSomethingWith(data);  
});  
  
doSomethingElse();
```



**WIN ✓**

No need to wait for the disk,  
do something else meanwhile!

# The Present



# CommonJS Modules

hello.js

```
exports.world = function() {  
  return 'Hello World';  
};
```

main.js

```
var hello = require('./hello');  
var sys = require('sys');  
sys.puts(hello.world());
```

```
$ node main.js  
Hello World
```

# Child processes

child.js

```
var child = process.createChildProcess('sh',  
  ['-c', 'echo hello; sleep 1; echo world;']);  
child.addListener('output', function (chunk) {  
  p(chunk);  
});
```

```
$ node child.js  
"hello\n"  
# 1 sec delay  
"world\n"  
null
```

# Http Server

```
var http = require('http');
http.createServer(function(req, res) {
  setTimeout(function() {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.sendBody('Thanks for waiting!');
    res.finish();
  }, 1000);
}).listen(4000);
```

```
$ curl localhost:4000
# 1 sec delay
Thanks for waiting!
```

# Tcp Server

```
var tcp = require('tcp');
tcp.createServer(function(socket) {
  socket.addListener('connect', function() {
    socket.send("Hi, How Are You?\n> ");
  });
  socket.addListener('receive', function(data) {
    socket.send(data);
  });
}).listen(4000);
```

```
$ nc localhost 4000
Hi, How Are You?
> Great!
Great!
```

# DNS

dns.js

```
var dns = require('dns');  
dns.resolve4('nodejs.org')  
  .addCallback(function(r) {  
    p(r);  
  });
```

```
$ node dns.js  
[  
  "97.107.132.72"  
]
```

# Watch File

watch.js

```
process.watchFile(__filename, function() {  
  puts('You changed me!');  
  process.exit();  
});
```

```
$ node watch.js  
# edit watch.js  
You changed me!
```

# ECMAScript 5

- Getters / setters

```
var a = {};  
a.__defineGetter__('foo', function() {  
    return 'bar';  
});  
puts(a.foo);
```

- Array: filter, forEach, reduce, etc.
- JSON.stringify(), JSON.parse()

# There is only 1 thread

```
file.read('file.txt', function(data) {  
  // Will never fire  
});  
  
while (true) {  
  // this blocks the entire process  
}
```

Good for conceptual simplicity  
Bad for CPU-bound algorithms



# The Future


# Web workers

- Multiple node processes that do interprocess communication
- CPU-bound algorithms can run separately
- Multiple CPU cores can be used efficiently

# Move C/C++ stuff to JS

- Simplifies the code base
- Makes contributions easier
- Low-level bindings = more flexibility

# Better Socket Support

- Support for unix sockets, `socketpair()`, `pipe()`
- Pass sockets between processes  load balance requests between web workers
- Unified socket streaming interfaces

# Even more ...

- Connecting streams

```
var f = file.writeStream('/tmp/x.txt');  
connect(req.body, f);
```

- Http parser bindings
- No memcpy() for http requests

+ hot code reloading!

# Suitable Applications

- Web frameworks
- Real time
- Crawlers

# More Applications

- Process monitoring
- File uploading
- Streaming

**Demo Time!**



# Http Chat in 14 LoC

```
var
  http = require('http'),
  messages = [];

http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type' : 'text/plain'});
  if (req.url == '/') {
    res.sendBody(messages.join("\n"));
  } else if (req.url !== '/favicon.ico') {
    messages.push(decodeURIComponent(req.url.substr(1)));
    res.sendBody('ok!');
  }
  res.finish();
}).listen(4000);
```

The chat room:

<http://debuggable.com:4000/>

Send a message:

<http://debuggable.com:4000/<msg>>

# Questions?



@felixge



<http://debuggable.com/>

**Bonus Slides!**

# Dirty



NoSQL for the little man!

# Dirty



JavaScript Views  
Disk Persistence



Dirty



Memory Store  
Speed > Safety

# Dirty Hello World

hello.js

```
var
  Dirty = require('../lib/dirty').Dirty,
  posts = new Dirty('test.dirty');

posts.add({hello: 'dirty world!'});
posts.set('my-key', {looks: 'nice'});
```

```
$ node hello.js
$ cat test.dirty
{"hello":"dirty world!","_key":"3b8f86..."}
{"looks":"nice","_key":"my-key"}
```

# Reloading from Disk

hello.js

```
var
  Dirty = require('../lib/dirty').Dirty,
  posts = new Dirty('test.dirty');

posts.load()
  .addCallback(function() {
    p(posts.get('my-key'));
  });
```

```
$ node hello.js
{"looks": "nice", "_key": "my-key"}
```



# Filtering records

hello.js

```
var
  Dirty = require('../lib/dirty').Dirty,
  posts = new Dirty('test.dirty');

posts.load()
  .addCallback(function() {
    var docs = posts.filter(function(doc) {
      return ('hello' in doc);
    });
    p(docs);
  });
```

```
$ node hello.js
[{"hello": "dirty world!", "_key": "3b8f86..."}]
```

# Benchmarks

**Do your own!**

# My Results

- Set: 100k docs / sec
- Iterate: 33 million docs / sec
- Filter: 14 million docs / sec

(on my laptop - your milage may vary)

# Use Cases

- Small projects (db < memory)
- Rapid prototyping
- Add HTTP/TCP interface and scale

<http://github.com/felixge/node-dirty>

(or google for “dirty felixge”)